

Step 1: HTTPS & Certificate Test

Test HTTPS Connection <https://127.0.0.1:8000/api/posts/>

The screenshot shows the Postman interface with a collection named "Connectly API Security Tests". The "Test HTTPS Connection" step is selected. A GET request is made to `((base_url))/posts/` with the URL set to `https://127.0.0.1:8000/api`. The response status is 200 OK, and the JSON body contains two posts with their respective details and comments.

```
1 [ 2   { 3     "id": 1, 4       "content": "Another test post", 5       "author": 1, 6       "created_at": "2026-01-17T16:24:38.462247Z", 7       "comments": [ 8         "Comment by newuser on Post 1", 9         "Comment by newuser on Post 1", 10        "Comment by newuser on Post 1", 11        "Comment by newuser on Post 1" 12     ], 13   }, 14   { 15     "id": 2, 16       "content": "First post by author_user", 17       "author": 6, 18       "created_at": "2026-02-06T23:55:47.918447Z", 19       "comments": [ 20         "Comment by regular_user on Post 2", 21         "Comment by admin_user on Post 2" 22     ], 23   } ]
```

Step 2: Password Encryption Tests

Register New User (Hashed Password) <https://127.0.0.1:8000/api/register/>

The screenshot shows the Postman interface with the same collection. The "POST Register New User (Hashed P..." step is selected. A POST request is made to `((base_url))/register/` with the URL set to `https://127.0.0.1:8000/api`. The response status is 201 Created, and the JSON body indicates the user was registered successfully.

```
1 { 2   "message": "User registered successfully.", 3   "username": "postman_user_1770485943" 4 }
```

Login with Valid Credentials https://127.0.0.1:8000/api/login/

The screenshot shows the Postman interface with a collection named "Connectify API Security Tests". A test step titled "POST Login with Valid Credentials" is selected. The request URL is set to `((base_url))/login` with the base URL being `https://127.0.0.1:8000/api`. The response status is 200 OK, and the JSON body contains the message "Authentication successful!" and the username "admin_user".

Login with Invalid Credentials https://127.0.0.1:8000/api/login/

The screenshot shows the Postman interface with the same collection and test step. The response status is 401 Unauthorized, and the JSON body indicates an error: "error": "Invalid credentials."

Login Missing Credentials https://127.0.0.1:8000/api/login/

The screenshot shows the Postman interface with the same collection and test step. The response status is 400 Bad Request, and the JSON body indicates an error: "error": "Username and password are required."

Step 3: RBAC & Permission Tests

Access Protected Endpoint (No Token) <https://127.0.0.1:8000/api/protected/>

The screenshot shows the API security test interface with the following details:

- Test Environment:** Connectify API Security Tests / Step 3: RBAC & Permission Tests / Access Protected Endpoint (No Token)
- Method:** GET
- URL:** `((base_url)) /protected/`
- Params:** https://127.0.0.1:8000/api
- Query Params:** Collection
- Body:** JSON
- Response Status:** 401 Unauthorized
- Response Headers:** 28 ms, 499 B
- Response Body:**

```
1 [ 2 | "detail": "Authentication credentials were not provided." 3 ]
```

Access Protected Endpoint (With Token) <https://127.0.0.1:8000/api/protected/>

The screenshot shows the API security test interface with the following details:

- Test Environment:** Connectify API Security Tests / Step 3: RBAC & Permission Tests / Access Protected Endpoint (With Token)
- Method:** GET
- URL:** `((base_url)) /protected/`
- Params:** https://127.0.0.1:8000/api
- Query Params:** Collection
- Body:** JSON
- Response Status:** 200 OK
- Response Headers:** 14 ms, 434 B
- Response Body:**

```
1 [ 2 | "message": "Authenticated!" 3 ]
```

Access Own Post (Author) <https://127.0.0.1:8000/api/posts/4/>

The screenshot shows the API security test interface with the following details:

- Test Environment:** Connectify API Security Tests / Step 3: RBAC & Permission Tests / Access Own Post (Author)
- Method:** GET
- URL:** `((base_url)) /posts/4/`
- Params:** https://127.0.0.1:8000/api
- Query Params:** Collection
- Body:** JSON
- Response Status:** 200 OK
- Response Headers:** 16 ms, 438 B
- Response Body:**

```
1 [ 2 | "content": "Post by admin_user" 3 ]
```

Access Others Post (Non-Author) <https://127.0.0.1:8000/api/posts/4/>

The screenshot shows the Postman interface with a collection named "Connecty API Security Tests". A step titled "Step 3: RBAC & Permission Tests" contains a "GET Access Others Post (Non-Author)" test. The request URL is `(base_url)/posts/4/`. The response status is 403 Forbidden, with the body containing the message: "You do not have permission to perform this action."

Step 4: CRUD Operations - Users Get All Users <https://127.0.0.1:8000/api/users/>

The screenshot shows the Postman interface with the same collection. A step titled "Step 4: CRUD Operations - Users" contains a "GET Get All Users" test. The request URL is `(base_url)/users/`. The response status is 200 OK, and the body shows a JSON array of three user objects.

```
[{"id": 1, "username": "newuser", "email": "newuser@example.com", "created_at": "2026-01-17T16:04:50.072461Z"}, {"id": 2, "username": "Jaf", "email": "grenigiajaf@example.com", "created_at": "2026-01-17T16:12:14.941585Z"}, {"id": 3, "username": "Ralph", "email": "ralphnocon@example.com", "created_at": "2026-01-22T13:48:01.167486Z"}]
```

Create User <https://127.0.0.1:8000/api/users/>

The screenshot shows the Postman interface with the same collection. A step titled "Step 5: CRUD Operations - Posts" contains a "POST Create User" test. The request URL is `(base_url)/users/`. The response status is 201 Created, and the body shows a newly created user object with id 10.

```
{"id": 10, "username": "crud_test_user_1770486617", "email": "crudtest1770486617@example.com", "created_at": "2026-02-07T17:50:16.776246Z"}
```

Create User - Validation Error <https://127.0.0.1:8000/api/users/>

The screenshot shows the Postman interface with a collection named "Immaculate De Guzman's Worksp..." containing various test steps. The current step is "Step 4: CRUD Operations - Users / Create User - Validation Error". The request method is POST, and the URL is `((base_url))/users/`. The "Body" tab shows a JSON payload with two fields: "username" and "email". Both fields have validation errors: "username" is required and must be a valid email address. The response status is 400 Bad Request, and the response body contains the validation errors.

```
1 [ 2   "username": [ 3     "This field is required." 4   ], 5   "email": [ 6     "Enter a valid email address." 7   ] 8 ]
```

Step 5: CRUD Operations - Posts

Get All Posts <https://127.0.0.1:8000/api/posts/>

The screenshot shows the Postman interface with a collection named "Immaculate De Guzman's Worksp..." containing various test steps. The current step is "Step 5: CRUD Operations - Posts / Get All Posts". The request method is GET, and the URL is `((base_url))/posts/`. The "Body" tab shows the response body, which is a list of posts. Each post includes an ID, content, author, creation timestamp, and a list of comments.

```
1 [ 2   { 3     "id": 1, 4     "content": "Another test post", 5     "author": "another_user", 6     "created_at": "2026-01-17T16:24:38.452247Z", 7     "comments": [ 8       "Comment by newuser on Post 1", 9       "Comment by newuser on Post 1", 10      "Comment by newuser on Post 1", 11      "Comment by newuser on Post 1" 12     ] 13   }, 14   { 15     "id": 2, 16     "content": "first post by author_user", 17     "author": 6, 18     "created_at": "2026-02-05T23:55:47.918447Z", 19     "comments": [ 20       "Comment by regular_user on Post 2", 21       "Comment by admin_user on Post 2" 22     ] 23   }, 24 ]
```

Create Post <https://127.0.0.1:8000/api/posts/>

The screenshot shows the Postman interface with a successful API call. The URL is `https://127.0.0.1:8000/api/posts/`. The response status is `201 Created` with a response time of `67 ms` and a size of `569 B`. The response body is:

```
1 {
2     "id": 1,
3     "content": "This is a test post created via Postman at 1770406799",
4     "author": 1,
5     "created_at": "2025-02-07T17:53:19.268939Z",
6     "comments": []
7 }
```

Create Post - Invalid Author <https://127.0.0.1:8000/api/posts/>

The screenshot shows the Postman interface with an invalid author ID. The URL is `https://127.0.0.1:8000/api/posts/`. The response status is `400 Bad Request` with a response time of `25 ms` and a size of `489 B`. The response body is:

```
1 {
2     "author": [
3         "Invalid pk \"999999\" - object does not exist."
4     ]
5 }
```

Step 6: CRUD Operations - Comments

Get All Comments <https://127.0.0.1:8000/api/comments/>

The screenshot shows the Postman interface with the following details:

- Collection:** Step 6: CRUD Operations - Comments
- Method:** GET
- URL:** {{base_url}}/comments/
- Body:** (empty)
- Headers:** (empty)
- Test Results:** 200 OK, 17 ms, 1.06 KB
- Response Body (JSON):**

```
[{"id": 1, "text": "This is a comment.", "author": 1, "post": 1, "created_at": "2026-01-24T17:43:14.437943Z"}, {"id": 2, "text": "Great post!", "author": 6, "post": 2, "created_at": "2026-02-05T23:55:47.965288Z"}, {"id": 3, "text": "Thanks for sharing", "author": 4, "post": 2, "created_at": "2026-02-05T23:55:47.978145Z"}]
```

Create Comment <https://127.0.0.1:8000/api/comments/>

The screenshot shows the Postman interface with the following details:

- Collection:** Step 6: CRUD Operations - Comments
- Method:** POST
- URL:** {{base_url}}/comments/
- Body:** (empty)
- Headers:** (empty)
- Test Results:** 201 Created, 76 ms, 549 B
- Response Body (JSON):**

```
{"id": 7, "text": "This is a test comment created via Postman", "author": 1, "post": 1, "created_at": "2026-02-07T17:57:32.182865Z"}
```

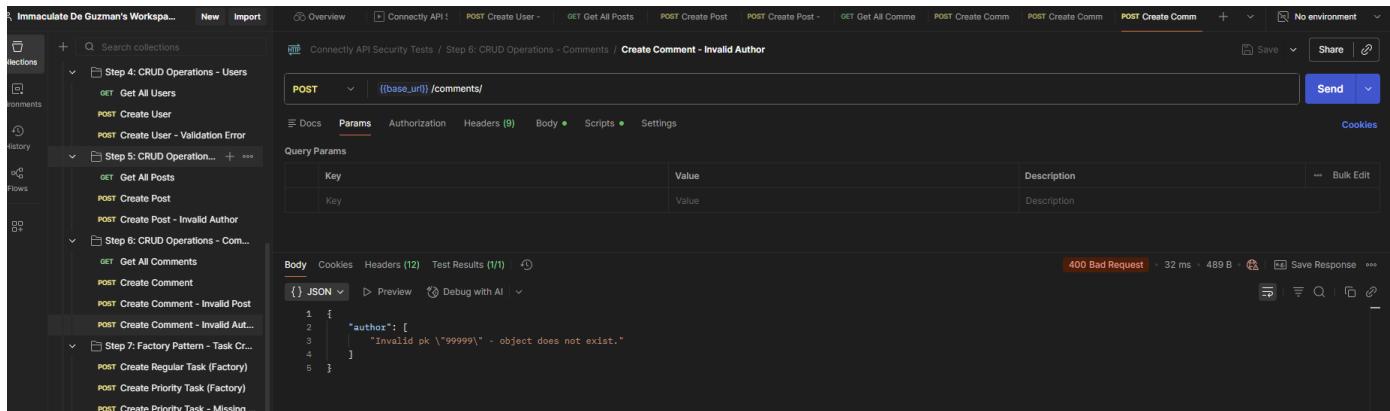
Create Comment - Invalid Post <https://127.0.0.1:8000/api/comments/>

The screenshot shows the Postman interface with the following details:

- Collection:** Step 6: CRUD Operations - Comments
- Method:** POST
- URL:** {{base_url}}/comments/
- Body:** (empty)
- Headers:** (empty)
- Test Results:** 400 Bad Request, 24 ms, 487 B
- Response Body (JSON):**

```
{"post": [{"post": "99999", "error": "Invalid pk \"99999\" - object does not exist."}]}
```

Create Comment - Invalid Author <https://127.0.0.1:8000/api/comments/>



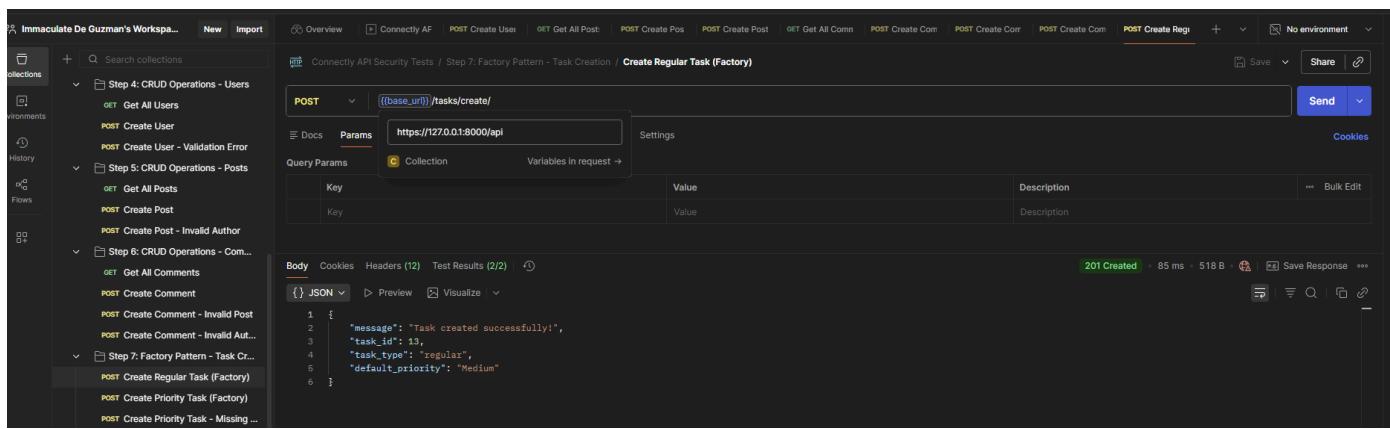
The screenshot shows the Postman interface with a collection named "Step 4: CRUD Operations - Users". A test step titled "Create Comment - Invalid Author" is selected. The request URL is `((base_url))/comments/`. The response status is 400 Bad Request, and the body contains the JSON object:

```
{ "author": [ "Invalid pk \"99999\" - object does not exist." ] }
```

.

Step 7: Factory Pattern - Task Creation

Create Regular Task (Factory) <https://127.0.0.1:8000/api/tasks/create/>

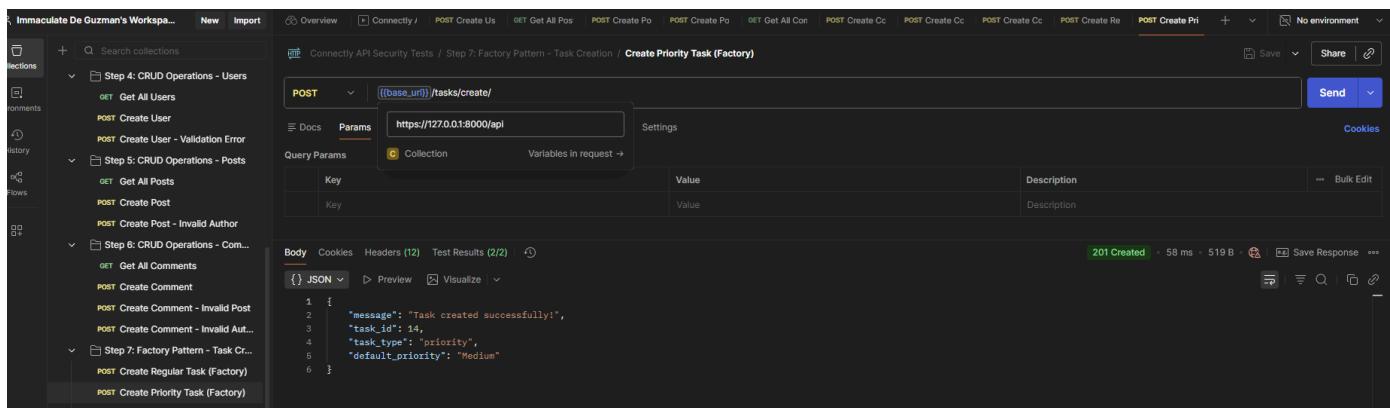


The screenshot shows the Postman interface with a collection named "Step 4: CRUD Operations - Users". A test step titled "Create Regular Task (Factory)" is selected. The request URL is `((base_url))/tasks/create/`. The response status is 201 Created, and the body contains the JSON object:

```
{ "message": "Task created successfully!", "task_id": 13, "task_type": "regular", "default_priority": "Medium" }
```

.

Create Priority Task (Factory) <https://127.0.0.1:8000/api/tasks/create/>



The screenshot shows the Postman interface with a collection named "Step 4: CRUD Operations - Users". A test step titled "Create Priority Task (Factory)" is selected. The request URL is `((base_url))/tasks/create/`. The response status is 201 Created, and the body contains the JSON object:

```
{ "message": "Task created successfully!", "task_id": 14, "task_type": "priority", "default_priority": "Medium" }
```

.

Create Priority Task - Missing Metadata (Factory Validation)

<https://127.0.0.1:8000/api/tasks/create/>

The screenshot shows the Connectly API Security Tests interface. The left sidebar lists various collections and steps. The main area shows a POST request to `((base_url))/tasks/create/`. The 'Body' tab displays a JSON payload with a single key 'error': "Priority tasks require 'priority_level' in metadata". The status bar at the bottom indicates a 400 Bad Request response.

Create Recurring Task (Factory) <https://127.0.0.1:8000/api/tasks/create/>

The screenshot shows the Connectly API Security Tests interface. The left sidebar lists various collections and steps. The main area shows a POST request to `https://127.0.0.1:8000/api/tasks/create/`. The 'Body' tab displays a JSON payload with keys 'message', 'task_id', 'task_type', and 'default_priority'. The status bar at the bottom indicates a 201 Created response.

Create Recurring Task - Missing Metadata (Factory Validation)

<https://127.0.0.1:8000/api/tasks/create/>

The screenshot shows the Connectly API Security Tests interface. The left sidebar lists various collections and steps. The main area shows a POST request to `((base_url))/tasks/create/`. The 'Body' tab displays a JSON payload with a single key 'error': "Recurring tasks require 'frequency' in metadata". The status bar at the bottom indicates a 400 Bad Request response.

Create Task - Invalid Type (Factory Validation) <https://127.0.0.1:8000/api/tasks/create/>

The screenshot shows the Postman interface with a collection named "Immaculate De Guzman's Worksp...". The "Step 7: Factory Pattern - Task Cr..." section contains a POST request to `(base_url)/tasks/create/`. The response status is 400 Bad Request, and the JSON body shows the error message: `"error": "Invalid task type. Must be one of: ['regular', 'priority', 'recurring']"`.

Create Task - Invalid User (Relational Validation) <https://127.0.0.1:8000/api/tasks/create/>

The screenshot shows the Postman interface with the same collection. The "Step 7: Factory Pattern - Task Cr..." section contains a POST request to `(base_url)/tasks/create/`. The response status is 400 Bad Request, and the JSON body shows the error message: `"error": "Assigned user not found."`.

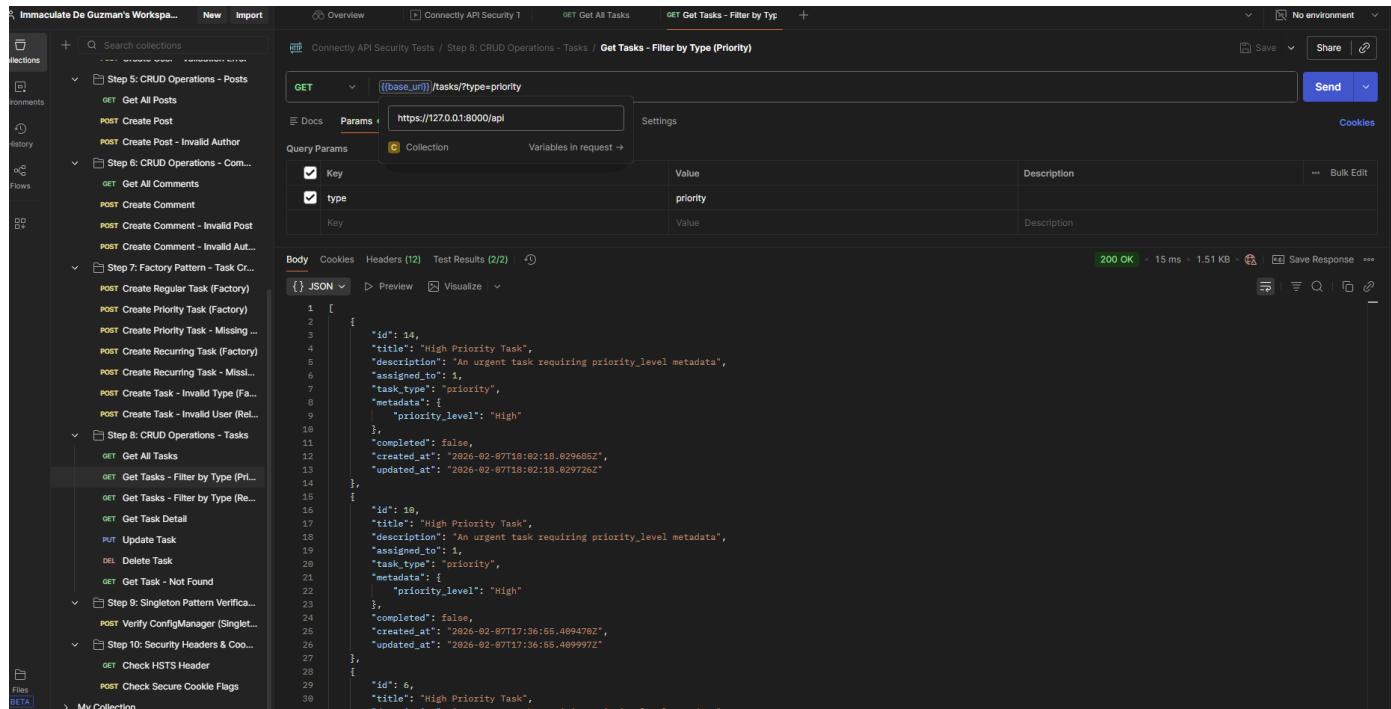
Step 8: CRUD Operations - Tasks

Get All Tasks <https://127.0.0.1:8000/api/tasks/>

The screenshot shows the Postman interface with the collection expanded. The "Step 8: CRUD Operations - Tasks" section contains a GET request to `(base_url)/tasks/`. The response status is 200 OK, and the JSON body lists three tasks:

```
[{"id": 15, "title": "Weekly Status Report", "description": "A recurring task that runs weekly", "assigned_to": 1, "task_type": "recurring", "metadata": {"frequency": "weekly"}, "completed": false, "created_at": "2026-02-07T18:04:17.292575Z", "updated_at": "2026-02-07T18:04:17.292512Z"}, {"id": 14, "title": "High Priority Task", "description": "An urgent task requiring priority_level metadata", "assigned_to": 1, "task_type": "priority", "metadata": {"priority_level": "High"}, "completed": false, "created_at": "2026-02-07T18:02:18.829685Z", "updated_at": "2026-02-07T18:02:18.829726Z"}, {"id": 13, "title": "Regular Test Task", "description": "A regular task created via Postman to test Factory Pattern", "assigned_to": 1}, ]
```

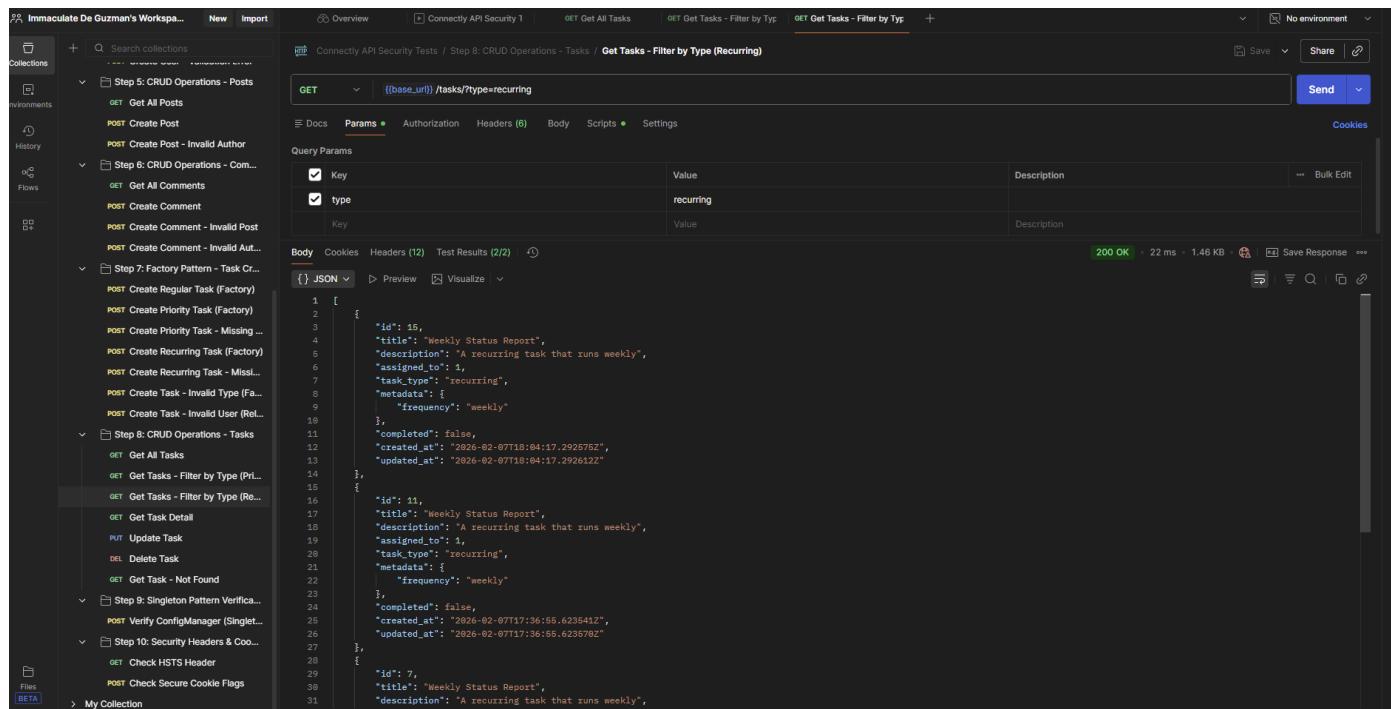
Get Tasks - Filter by Type (Priority) <https://127.0.0.1:8000/api/tasks/?type=priority>



The screenshot shows the Postman interface with a collection named "Step 8: CRUD Operations - Tasks". A specific test step "GET Get Tasks - Filter by Type (Priority)" is selected. The request URL is set to `https://127.0.0.1:8000/api/tasks/?type=priority`. The response body is displayed as JSON, showing two task objects:

```
[{"id": 14, "title": "High Priority Task", "description": "An urgent task requiring priority_level metadata", "assigned_to": 1, "task_type": "priority", "metadata": {"priority_level": "High"}, "completed": false, "created_at": "2026-02-07T18:02:18.029686Z", "updated_at": "2026-02-07T18:02:18.029726Z"}, {"id": 10, "title": "High Priority Task", "description": "An urgent task requiring priority_level metadata", "assigned_to": 1, "task_type": "priority", "metadata": {"priority_level": "High"}, "completed": false, "created_at": "2026-02-07T17:36:55.409470Z", "updated_at": "2026-02-07T17:36:55.409997Z"}]
```

Get Tasks - Filter by Type (Recurring) <https://127.0.0.1:8000/api/tasks/?type=recurring>



The screenshot shows the Postman interface with the same collection and test step as the previous screenshot. The request URL is now set to `https://127.0.0.1:8000/api/tasks/?type=recurring`. The response body shows two task objects:

```
[{"id": 16, "title": "Weekly Status Report", "description": "A recurring task that runs weekly", "assigned_to": 1, "task_type": "recurring", "metadata": {"frequency": "weekly"}, "completed": false, "created_at": "2026-02-07T18:04:17.292875Z", "updated_at": "2026-02-07T18:04:17.292612Z"}, {"id": 11, "title": "Weekly Status Report", "description": "A recurring task that runs weekly", "assigned_to": 1, "task_type": "recurring", "metadata": {"frequency": "weekly"}, "completed": false, "created_at": "2026-02-07T17:36:55.623541Z", "updated_at": "2026-02-07T17:36:55.623576Z"}]
```

Get Task Detail <https://127.0.0.1:8000/api/tasks/13>

The screenshot shows the Postman interface with the following details:

- Collection:** Immaculate De Guzman's Workspace
- Request Type:** GET
- URL:** {{base_url}}/tasks/((regular_task_id)) /
- Params:** https://127.0.0.1:8000/api
- Query Params:** Collection: regular_task_id
- Body:** JSON (Preview shows task details with id 13)
- Response:** 200 OK (16 ms, 696 B)

Update Task <https://127.0.0.1:8000/api/tasks/13>

The screenshot shows the Postman interface with the following details:

- Collection:** Immaculate De Guzman's Workspace
- Request Type:** PUT
- URL:** {{base_url}}/tasks/((regular_task_id)) /
- Params:** https://127.0.0.1:8000/api
- Query Params:** Collection: regular_task_id
- Body:** JSON (Preview shows updated task details with id 13 and title "Updated Task Title")
- Response:** 200 OK (63 ms, 696 B)

Delete Task <https://127.0.0.1:8000/api/tasks/13>

The screenshot shows the Postman interface with the following details:

- Collection:** Immaculate De Guzman's Workspace
- Request Type:** DELETE
- URL:** {{base_url}}/tasks/((regular_task_id)) /
- Params:** https://127.0.0.1:8000/api
- Query Params:** Collection: regular_task_id
- Body:** JSON (Preview shows message "Task deleted successfully.")
- Response:** 204 No Content (54 ms, 475 B)

Get Task - Not Found <https://127.0.0.1:8000/api/tasks/99999>

The screenshot shows the Postman interface with a collection named "Step 5: CRUD Operations - Posts". A specific test step is selected: "GET Get All Posts". The URL is set to `https://127.0.0.1:8000/api/tasks/99999`. The response status is 404 Not Found, and the body contains the JSON object `{"error": "Task not found."}`.

Step 9: Singleton Pattern Verification

Verify ConfigManager (Singleton) <https://127.0.0.1:8000/api/tasks/create/>

The screenshot shows the Postman interface with a collection named "Step 5: CRUD Operations - Posts". A specific test step is selected: "POST Create Post". The URL is set to `https://127.0.0.1:8000/api/tasks/create/`. The response status is 201 Created, and the body contains the JSON object `{"message": "Task created successfully!", "task_id": 16, "task_type": "regular", "default_priority": "Medium"}`.

Step 10: Security Headers & Cookies

Check HSTS Header <https://127.0.0.1:8000/api/posts/>

The screenshot shows the Postman interface with a collection named "Step 5: CRUD Operations - Posts". A specific test step is selected: "GET Get All Posts". The URL is set to `https://127.0.0.1:8000/api/posts/`. The response status is 200 OK, and the body contains a JSON array of posts, each with fields like id, content, author, created_at, and comments.

Check Secure Cookie Flags <https://127.0.0.1:8000/api/login/>

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Collections' (including 'Immaculate De Guzman's Worksp...'), 'Environments', 'History', and 'Flows'. The main workspace shows a collection named 'Step 5: CRUD Operations - Posts' containing several POST and GET requests. A specific POST request to 'https://127.0.0.1:8000/api/login/' is selected. The 'Params' tab shows 'base_url' set to 'https://127.0.0.1:8000/api'. The 'Body' tab is set to 'JSON' and contains the following JSON response:

```
1 {
2     "message": "Authentication successful!",
3     "username": "admin_user"
4 }
```

The status bar at the bottom indicates a '200 OK' response with '190 ms' and '473 B'.